**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**OPERATING SYSTEMS**

***Submitted by***

**PRANAV HEGDE (1BM22CS202)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**Apr-2024 to Aug-2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “OPERATING SYSTEMS – 23CS4PCOPS” carried out by **PRANAV HEGDE (1BM22CS202),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **OPERATING SYSTEMS - (23CS4PCOPS)** work prescribed for the said degree.

**Dr. Asha G R**                **Dr. Jyothi S Nayak**

Asst. Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Table Of Contents**

|  |  |  |
| --- | --- | --- |
| **Lab Program No.** | **Program Details** | **Page No.** |
| 1 | FCFS AND SJF (NON-PRE-EMPTIVE) | 3-8 |
| 2 | SJF (PREEMPTIVE) AND ROUND ROBIN | 9-15 |
| 3 | PRIORITY (PRE-EMPTIVE AND NON-PRE-EMPTIVE) | 16-22 |
| 4 | MULTI-LEVEL QUEUE | 23-26 |
| 5 | RATE-MONOTONIC, EARLIEST DEADLINE FIRST AND PROPORTIONAL | 27-34 |
| 6 | PRODUCER-CONSUMER PROBLEM | 35-38 |
| 7 | DINING-PHILOSOPHERS PROBLEM | 39-43 |
| 8 | BANKERS ALGORITHM(DEADLOCK AVOIDANCE AND DETECTION) | 44-46 |
| 9 | MEMORY ALLOCATION (FIRST FIT, BEST FIT AND WORST FIT) | 47-52 |
| 10 | PAGE REPLACEMENT(FIFO, OPTIMAL AND LRU) | 53-60 |

**Course Outcomes**

**CO1:** Apply the different concepts and functionalities of Operating System.

**CO2:** Analyze various Operating system strategies and techniques.

**CO3:** Demonstrate the different functionalities of Operating System.

**CO4:** Conduct practical experiments to implement the functionalities of Operating system.

**Lab Program – 1**

**Question:**

**Write a C program to simulate the following non-pre-emptive CPU**

**scheduling algorithm to find turnaround time and waiting time.**

**a) FCFS**

**b) SJF (Non-preemptive)**

**Code:**

1. **FCFS**

#include <stdio.h>

#define max 10

struct P{

int id;

int bt;

int at;

int wt;

int tat;

int rt;

int st;

int et;

int v;

};

void main(){

int n,ct=0;

float awt=0,atat=0,art=0,tp;

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

for(int i=0;i<n;i++){

printf("Enter Process %d ID: ",i+1);

scanf("%d",&p[i].id);

printf("Enter Burst Time and Arrival Time %d: ",i+1);

scanf("%d %d",&p[i].bt,&p[i].at);

}

for(int i=0;i<n;i++){

for(int j=i+1;j<n;j++){

if(p[i].at>p[j].at){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

for(int i=0;i<n;i++){

if(p[i].at<ct){

p[i].st=ct;

}

else{

p[i].st=p[i].at;

}

p[i].et=p[i].st+p[i].bt;

ct+=p[i].bt;

}

for(int i=0;i<n;i++){

p[i].tat=p[i].et-p[i].at;

p[i].wt=p[i].tat-p[i].bt;

p[i].rt=p[i].st-p[i].at;

}

printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){

printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);

awt+=p[i].wt;

atat+=p[i].tat;

art+=p[i].rt;

}

tp=(float)p[n-1].et/n;

printf("Average Waiting Time: %.2f\n",awt/n);

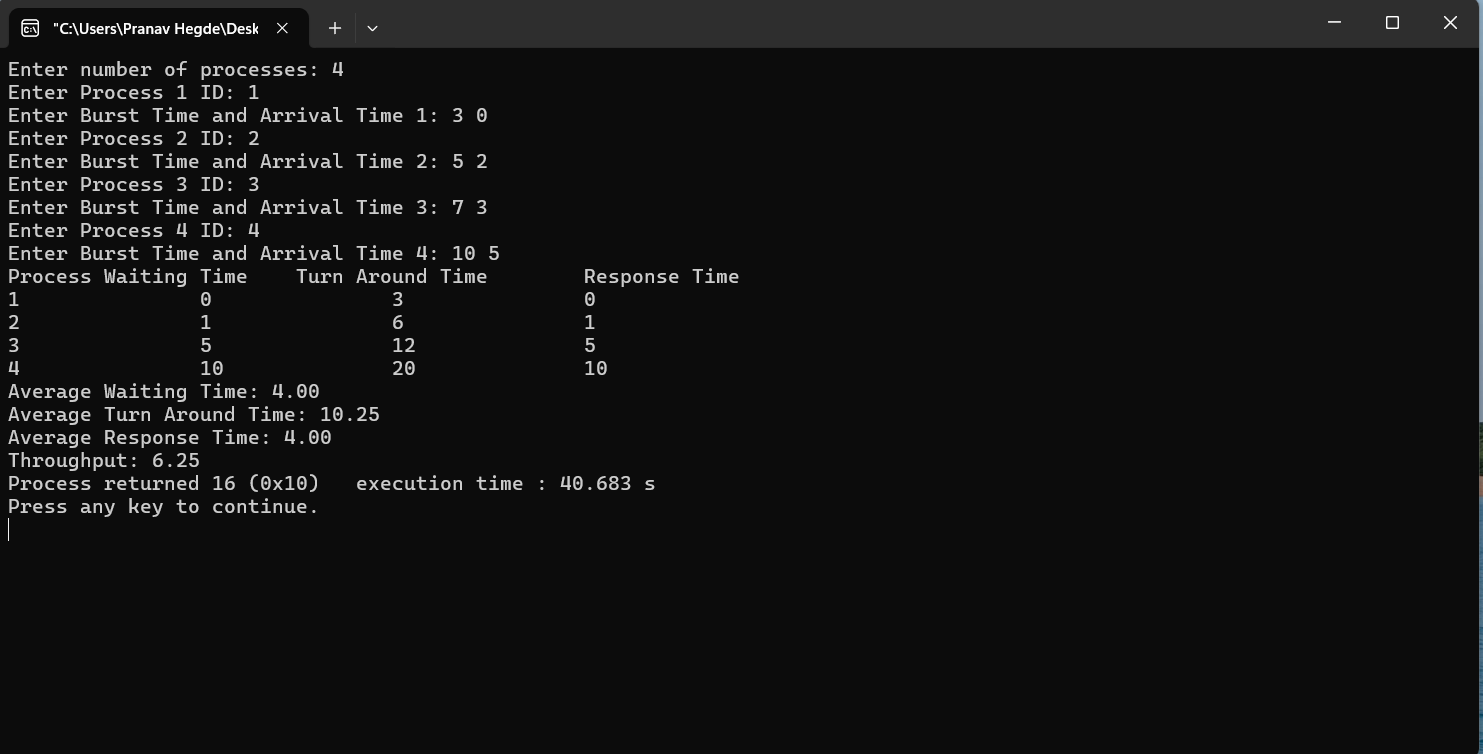
printf("Average Turn Around Time: %.2f\n",atat/n);

printf("Average Response Time: %.2f\n",art/n);

printf("Throughput: %.2f",tp);

}

**Output:**



**b) SJF (Non-preemptive)**

#include <stdio.h>

#include <limits.h>

struct P{

int id;

int bt;

int at;

int wt;

int tat;

int rt;

int st;

int et;

int v;

};

void main(){

int n,ct=0;

float awt=0,atat=0,art=0,tp;

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

for(int i=0;i<n;i++){

p[i].id=i+1;

p[i].v=0;

printf("Enter Burst Time and Arrival Time of P%d: ",i+1);

scanf("%d %d",&p[i].bt,&p[i].at);

}

for(int i=0;i<n;i++){

for(int j=i+1;j<n;j++){

if(p[i].at>p[j].at){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

if(p[i].at==p[j].at){

if(p[i].bt>p[j].bt){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

}

int cf=p[0].bt, min=INT\_MAX,m,count=0;

p[0].v=1;

while(count<n-1){

for(int i=0;i<n;i++){

if(p[i].at<=cf && p[i].v==0){

if(p[i].bt<min){

min=p[i].bt;

m=i;

}

}

}

p[m].v=1;

cf+=p[m].bt;

min=INT\_MAX;

temp=p[count+1];

p[count+1]=p[m];

p[m]=temp;

count++;

}

printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){

if(p[i].at<ct){

p[i].st=ct;

}

else{

p[i].st=p[i].at;

}

p[i].et=p[i].st+p[i].bt;

ct+=p[i].bt;

p[i].tat=p[i].et-p[i].at;

p[i].wt=p[i].tat-p[i].bt;

p[i].rt=p[i].st-p[i].at;

printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);

awt+=p[i].wt;

atat+=p[i].tat;

art+=p[i].rt;

}

tp=(float)p[n-1].et/n;

printf("Average Waiting Time: %.2f\n",awt/n);

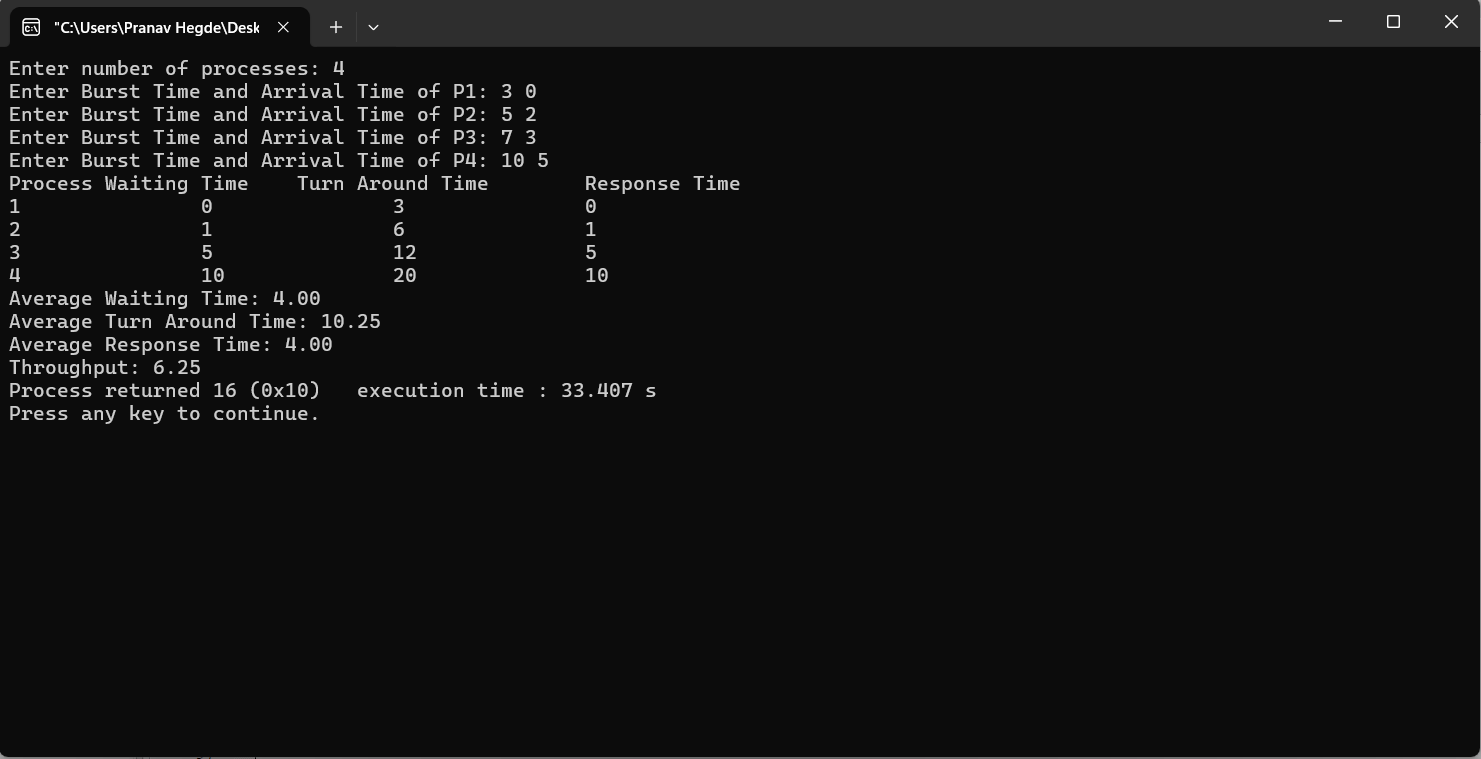
printf("Average Turn Around Time: %.2f\n",atat/n);

printf("Average Response Time: %.2f\n",art/n);

printf("Throughput: %.2f",tp);

}

**Output:**

****

**Lab Program - 2**

**Question:**

**Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.  
a) SJF (Preemptive)   
b) Round Robin Algorithm**

**Code:**

1. **SJF (Preemptive)**

#include <stdio.h>

#include <limits.h>

struct P{

int id;

int bt;

int at;

int wt;

int tat;

int rt;

int st;

int et;

int v;

int b;

};

void main(){

int n,ct=0;

float awt=0,atat=0,art=0,tp;

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

for(int i=0;i<n;i++){

p[i].id=i+1;

p[i].v=0;

p[i].st=-1;

printf("Enter Burst Time and Arrival Time of P%d: ",i+1);

scanf("%d %d",&p[i].bt,&p[i].at);

p[i].b=p[i].bt;

}

int cf=0, min=INT\_MAX,m=0,count=0;

while(count<n){

for(int i=0;i<n;i++){

if(p[i].at<=cf){

if(p[i].bt==min){

min=p[i].at>p[m].at?p[i].bt:p[m].bt;

}

if(p[i].bt<min && p[i].v!=1){

min=p[i].bt;

m=i;

}

}

}

p[m].bt-=1;

if(p[m].st<0)

p[m].st=cf;

cf+=1;

if(p[m].bt==0){

p[m].et=cf;

count++;

p[m].v=1;

}

min=INT\_MAX;

}

printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){

p[i].tat=p[i].et-p[i].at;

p[i].wt=p[i].tat-p[i].b;

p[i].rt=p[i].st-p[i].at;

printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);

awt+=p[i].wt;

atat+=p[i].tat;

art+=p[i].rt;

}

tp=(float)cf/n;

printf("Average Waiting Time: %.2f\n",awt/n);

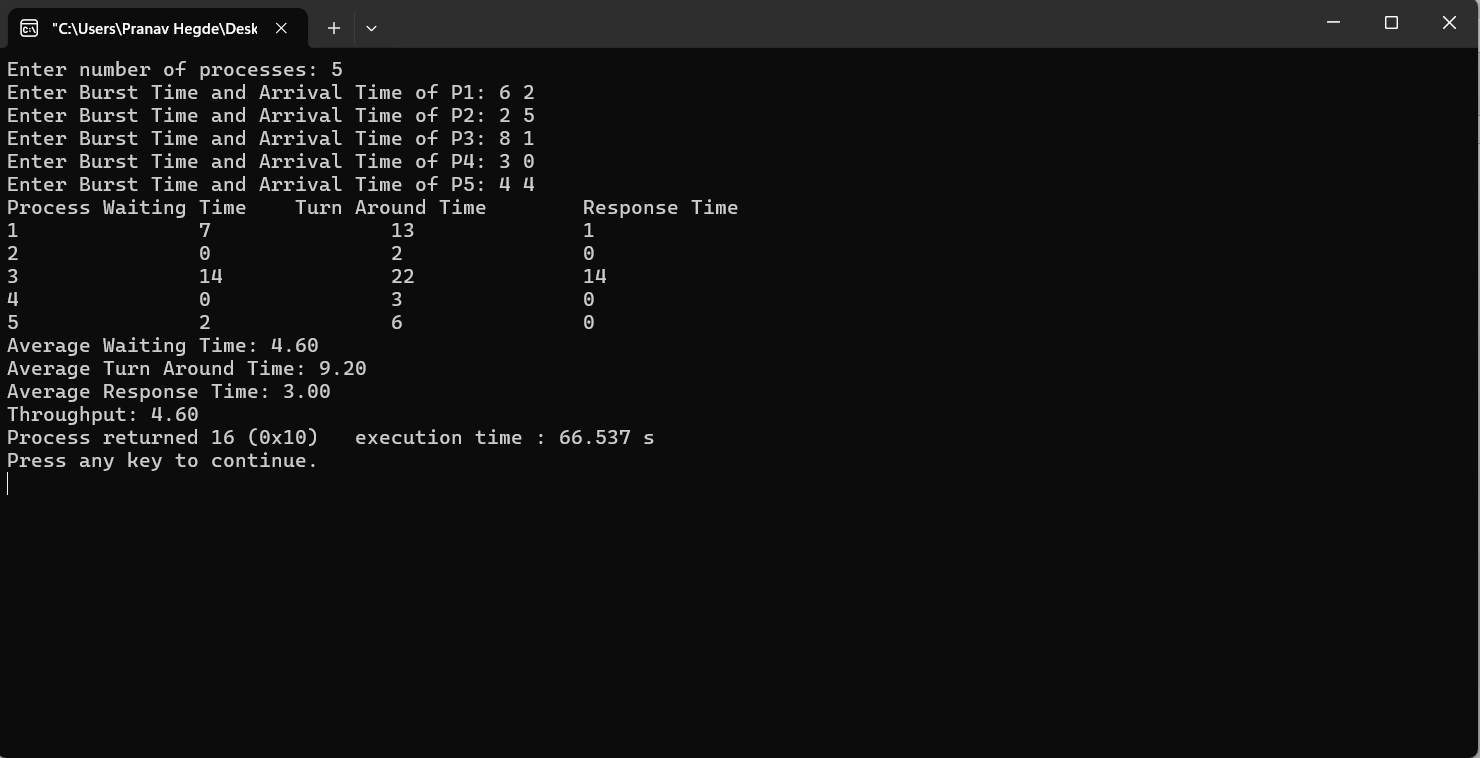
printf("Average Turn Around Time: %.2f\n",atat/n);

printf("Average Response Time: %.2f\n",art/n);

printf("Throughput: %.2f",tp);

}

**Output:**



1. **Round Robin**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PROCESSES 10

struct Process {

int id;

int at;

int bt;

int wt;

int tat;

int st;

int et;

int rt;

int vi;

int obt;

};

int main() {

struct Process p[MAX\_PROCESSES];

int n;

int total\_wt = 0;

int total\_tat = 0;

int total\_rt = 0;

int total\_time = 0;

int tq = 0;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter time quantum: ");

scanf("%d", &tq);

for (int i = 0; i < n; i++) {

p[i].id = i + 1;

printf("Enter arrival time and burst time for process %d: ", p[i].id);

scanf("%d %d", &p[i].at, &p[i].bt);

p[i].obt = p[i].bt;

p[i].st = -1;

p[i].et = -1;

p[i].vi = 0;

}

int count = 0;

int ind = 0;

int curr\_time = 0;

while(1){

int skipped = 0;

if(p[ind].at > curr\_time){

ind = (ind + 1) % n;

continue;

}

if(p[ind].st == -1){

p[ind].st = curr\_time;

}

if(p[ind].bt > tq){

p[ind].bt -= tq;

skipped = tq;

}

else if(p[ind].bt > 0){

skipped = p[ind].bt;

p[ind].bt = 0;

p[ind].et = curr\_time + skipped;

total\_time = curr\_time;

count++;

}

curr\_time += skipped;

ind = (ind + 1) % n;

if(count == n){

break;

}

}

for(int i = 0; i < n; i++){

p[i].tat = p[i].et - p[i].at;

p[i].wt = p[i].tat - p[i].obt;

p[i].rt = p[i].st - p[i].at;

total\_wt += p[i].wt;

total\_tat += p[i].tat;

total\_rt += p[i].rt;

}

printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\tResponse Time\n");

for (int i = 0; i < n; i++) {

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id, p[i].at, p[i].obt, p[i].wt, p[i].tat, p[i].rt);

}

printf("\nAverage Waiting Time: %.2f\n", (float)total\_wt / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_tat / n);

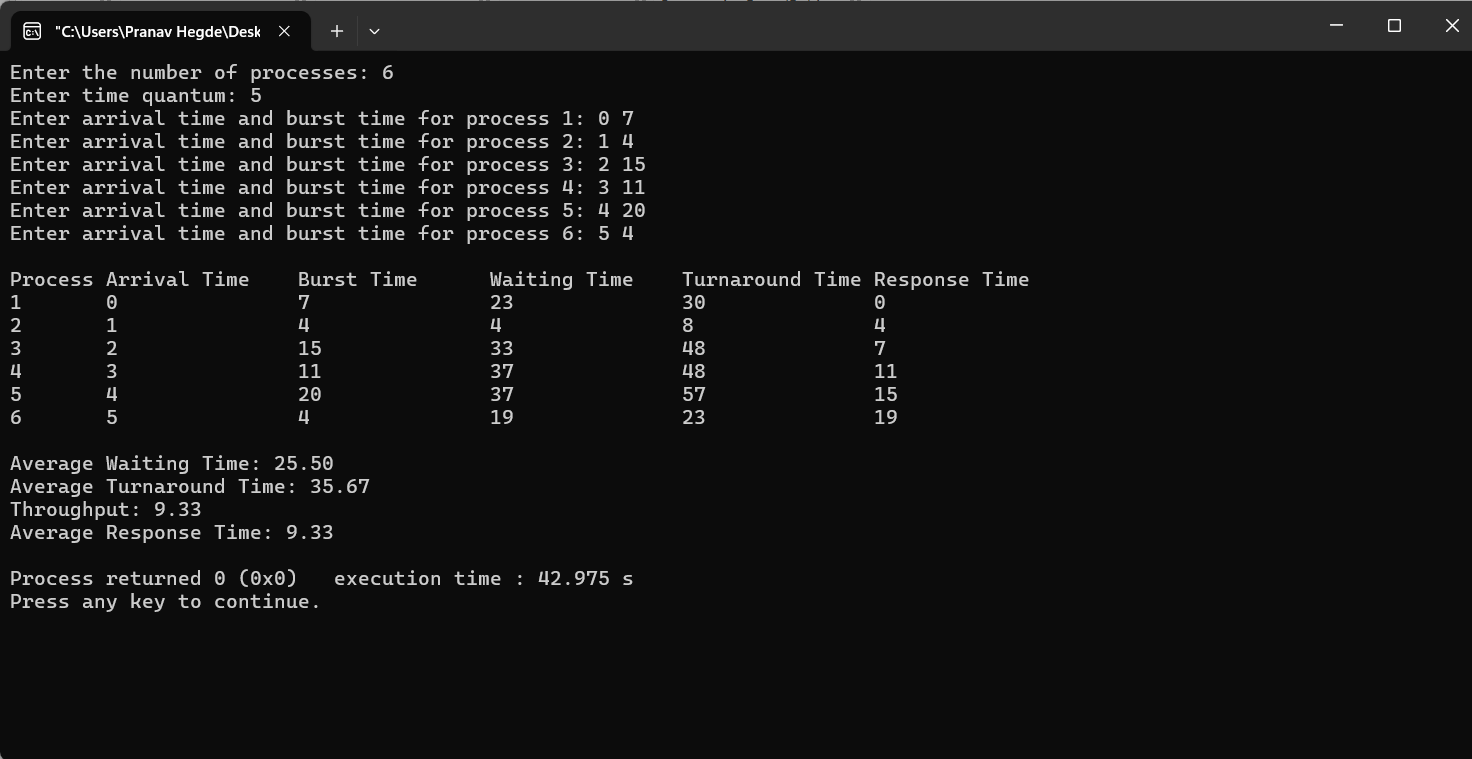
printf("Throughput: %.2f\n", (float)total\_time / n);

printf("Average Response Time: %.2f\n", (float)total\_rt / n);

return 0;

}

**Output:**



**Lab Program - 3**

**Question:**

**Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.**

**Priority (pre-emptive & Non-pre-emptive)** 

**Code:**

1. **Pre-emptive**

#include <stdio.h>

#include <limits.h>

struct P{

int id;

int bt;

int at;

int p;

int wt;

int tat;

int rt;

int st;

int et;

int v;

int b;

};

void main(){

int n,ct=0;

float awt=0,atat=0,art=0,tp;

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

for(int i=0;i<n;i++){

p[i].id=i+1;

p[i].v=0;

p[i].st=-1;

printf("Enter Burst Time, Arrival Time and Priority of P%d: ",i+1);

scanf("%d %d %d",&p[i].bt,&p[i].at,&p[i].p);

p[i].b=p[i].bt;

}

int cf=0, min=INT\_MAX,m=0,count=0;

while(count<n){

for(int i=0;i<n;i++){

if(p[i].at<=cf){

if(p[i].p<min && p[i].v!=1){

min=p[i].p;

m=i;

}

}

}

p[m].bt-=1;

if(p[m].st<0)

p[m].st=cf;

cf+=1;

if(p[m].bt==0){

p[m].et=cf;

count++;

p[m].v=1;

}

min=INT\_MAX;

}

printf("Process\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){

p[i].tat=p[i].et-p[i].at;

p[i].wt=p[i].tat-p[i].b;

p[i].rt=p[i].st-p[i].at;

printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);

awt+=p[i].wt;

atat+=p[i].tat;

art+=p[i].rt;

}

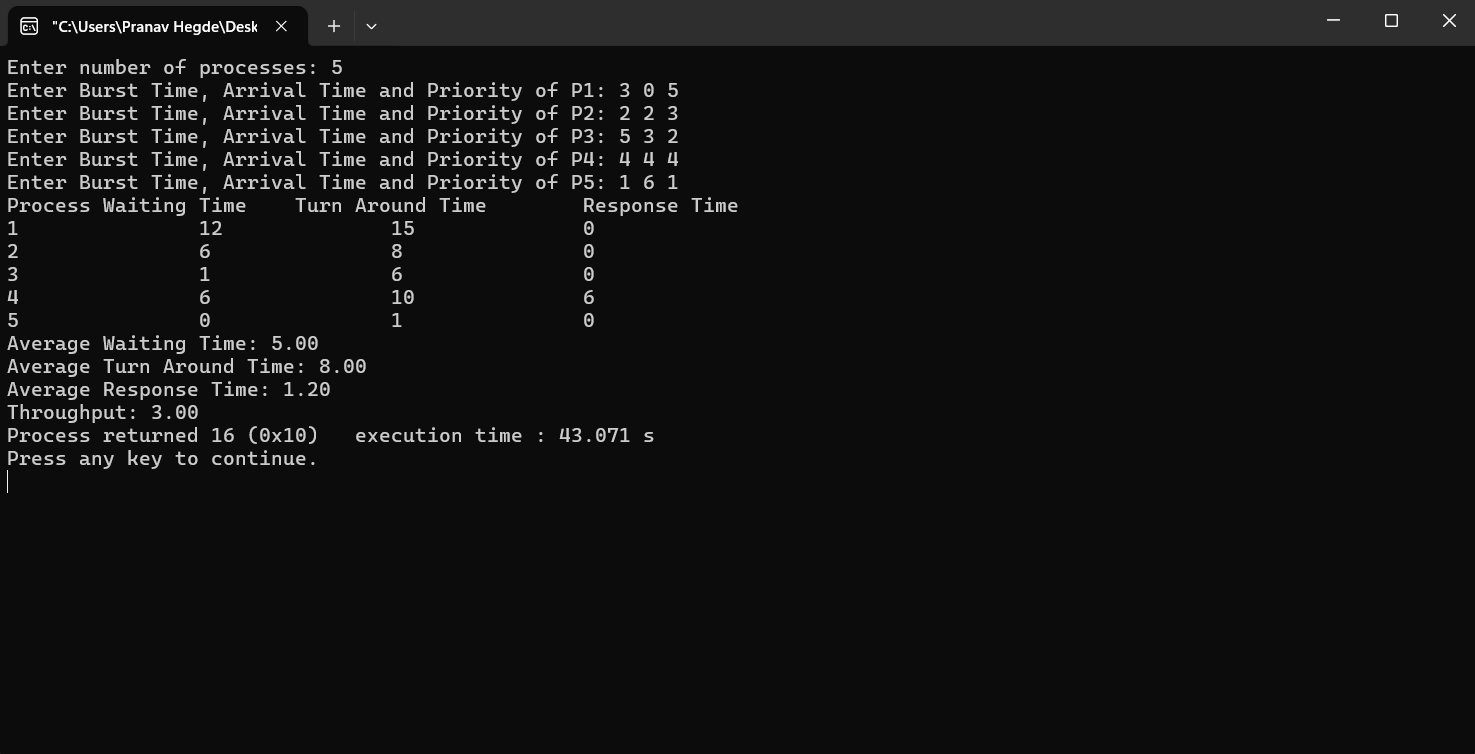
tp=(float)cf/n;

printf("Average Waiting Time: %.2f\n",awt/n);

printf("Average Turn Around Time: %.2f\n",atat/n);

printf("Average Response Time: %.2f\n",art/n);

printf("Throughput: %.2f",tp);

**Output:** 

1. **Non-Pre-emptive**

#include <stdio.h>

#include <limits.h>

struct P{

int id;

int bt;

int at;

int p;

int wt;

int tat;

int rt;

int st;

int et;

int v;

};

void main(){

int n,ct=0;

float awt=0,atat=0,art=0,tp;

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

for(int i=0;i<n;i++){

p[i].id=i+1;

p[i].v=0;

printf("Enter Burst Time, Arrival Time and Priority of P%d: ",i+1);

scanf("%d %d %d",&p[i].bt,&p[i].at,&p[i].p);

}

for(int i=0;i<n;i++){

for(int j=i+1;j<n;j++){

if(p[i].at>p[j].at){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

if(p[i].at==p[j].at){

if(p[i].p>p[j].p){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

}

int cf=p[0].bt, min=INT\_MAX,m,count=0;

p[0].v=1;

while(count<n-1){

for(int i=0;i<n;i++){

if(p[i].at<=cf && p[i].v==0){

if(p[i].p<min){

min=p[i].p;

m=i;

}

}

}

p[m].v=1;

cf+=p[m].bt;

min=INT\_MAX;

temp=p[count+1];

p[count+1]=p[m];

p[m]=temp;

count++;

}

printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){

if(p[i].at<ct){

p[i].st=ct;

}

else{

p[i].st=p[i].at;

}

p[i].et=p[i].st+p[i].bt;

ct+=p[i].bt;

p[i].tat=p[i].et-p[i].at;

p[i].wt=p[i].tat-p[i].bt;

p[i].rt=p[i].st-p[i].at;

printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);

awt+=p[i].wt;

atat+=p[i].tat;

art+=p[i].rt;

}

tp=(float)p[n-1].et/n;

printf("Average Waiting Time: %.2f\n",awt/n);

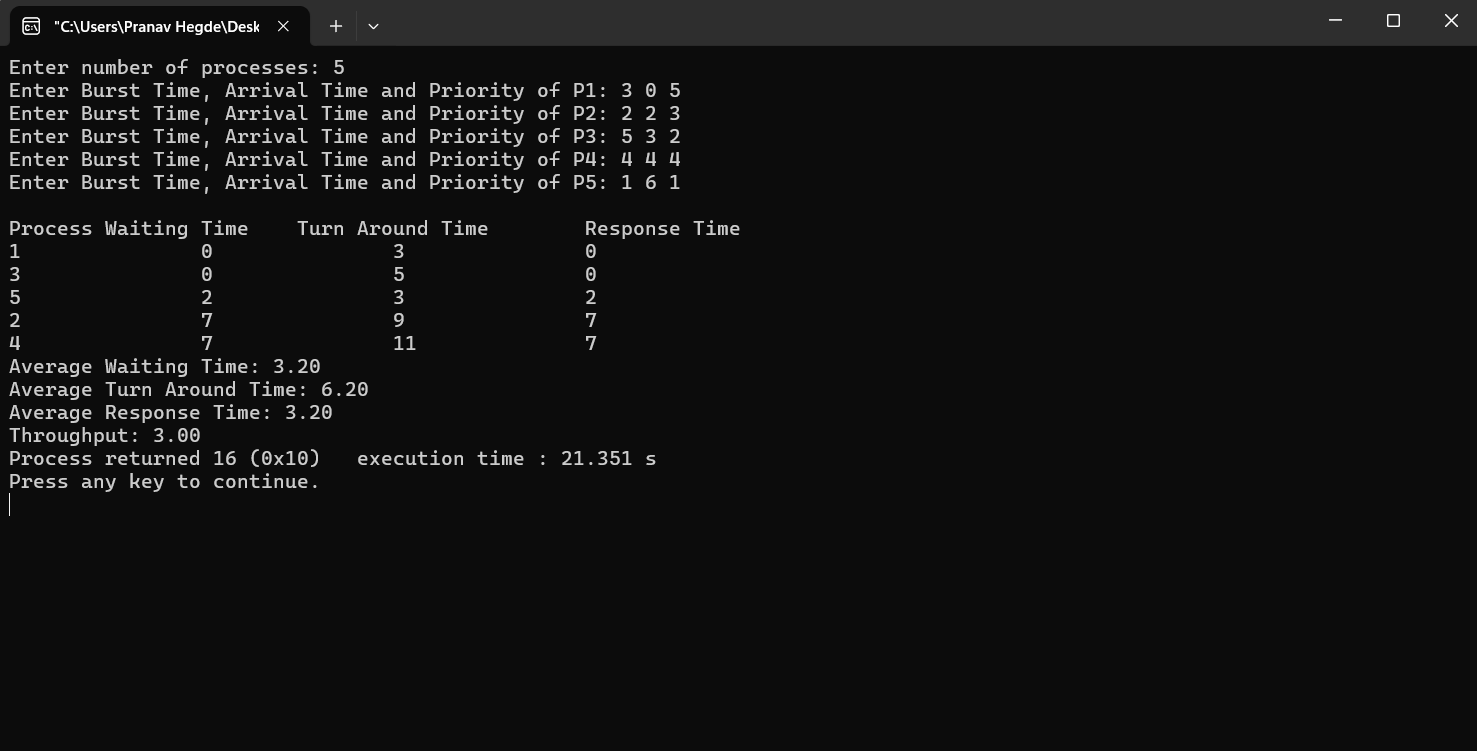
printf("Average Turn Around Time: %.2f\n",atat/n);

printf("Average Response Time: %.2f\n",art/n);

printf("Throughput: %.2f",tp);

}

**Output:**

****

**Lab Program - 4**

**Question:**

**Write a C program to simulate multi-level queue scheduling algorithm  
considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.**

**Code:**

#include <stdio.h>

#include <limits.h>

struct P{

int id;

int bt;

int at;

int q;

int wt;

int tat;

int rt;

int st;

int et;

int v;

};

void main(){

int n,ct=0;

float awt=0,atat=0,art=0,tp;

printf("Queue 1 is system process\nQueue 2 is User Process\n");

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

for(int i=0;i<n;i++){

p[i].id=i+1;

p[i].v=0;

printf("Enter Burst Time, Arrival Time and Queue of P%d: ",i+1);

scanf("%d %d %d",&p[i].bt,&p[i].at,&p[i].q);

}

for(int i=0;i<n;i++){

for(int j=i+1;j<n;j++){

if(p[i].at>p[j].at){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

if(p[i].at==p[j].at){

if(p[i].q>p[j].q){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

}

int cf=p[0].bt, min=INT\_MAX,m,count=0;

p[0].v=1;

while(count<n-1){

for(int i=0;i<n;i++){

if(p[i].at<=cf && p[i].v==0){

if(p[i].q<min){

min=p[i].q;

m=i;

}

}

}

p[m].v=1;

cf+=p[m].bt;

min=INT\_MAX;

temp=p[count+1];

p[count+1]=p[m];

p[m]=temp;

count++;

}

printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");

for(int i=0;i<n;i++){

if(p[i].at<ct){

p[i].st=ct;

}

else{

p[i].st=p[i].at;

}

p[i].et=p[i].st+p[i].bt;

ct+=p[i].bt;

p[i].tat=p[i].et-p[i].at;

p[i].wt=p[i].tat-p[i].bt;

p[i].rt=p[i].st-p[i].at;

printf("%d\t\t%d\t\t%d\t\t%d\n",p[i].id,p[i].wt,p[i].tat,p[i].rt);

awt+=p[i].wt;

atat+=p[i].tat;

art+=p[i].rt;

}

tp=(float)p[n-1].et/n;

printf("Average Waiting Time: %.2f\n",awt/n);

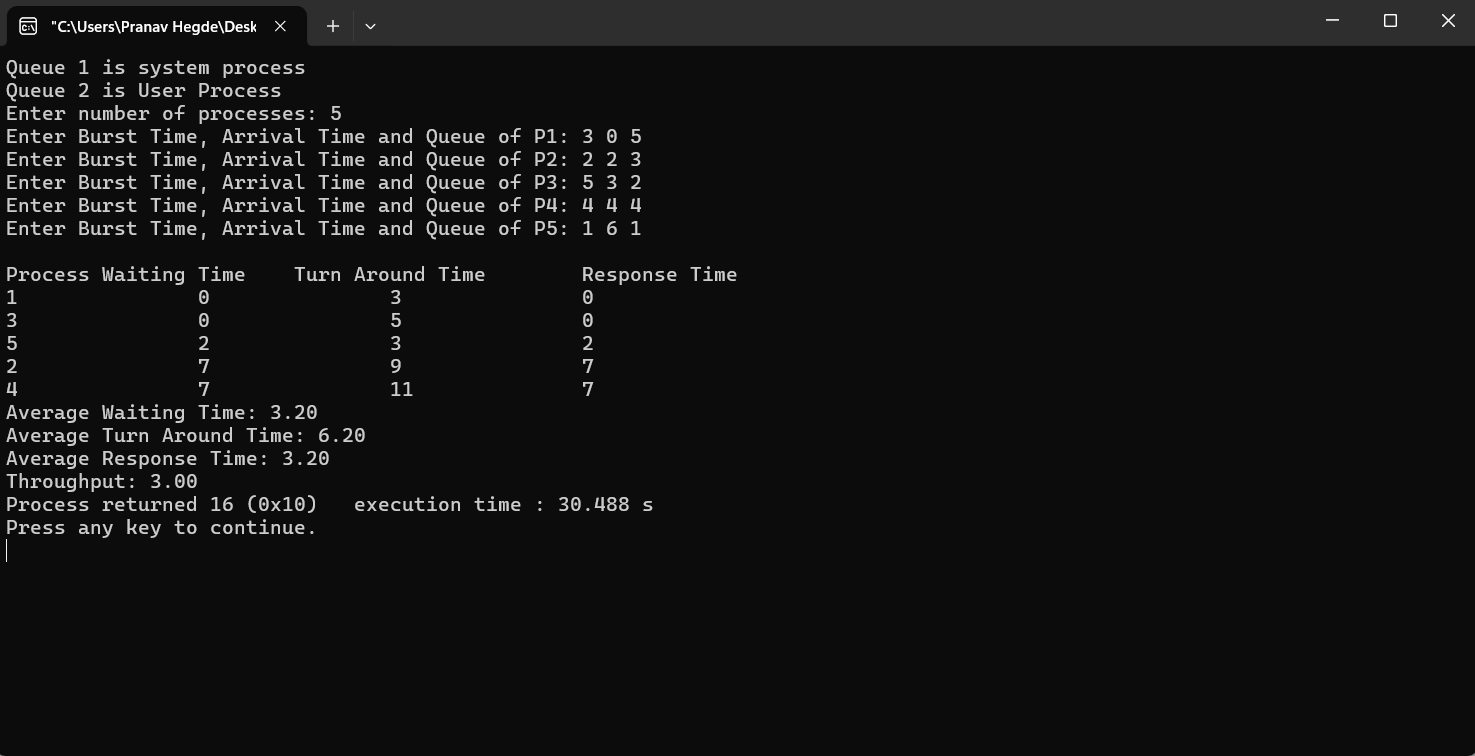
printf("Average Turn Around Time: %.2f\n",atat/n);

printf("Average Response Time: %.2f\n",art/n);

printf("Throughput: %.2f",tp);

}

**Output:**

****

**Lab Program - 5**

**Question:**

**Write a C program to simulate Real Time CPU Scheduling Algorithms:  
a) Rate- Monotonic  
b) Earliest Deadline First  
c) Proportional Scheduling**

**Code:**

1. **Rate-Monotonic**

#include <stdio.h>

#include <limits.h>

struct P{

int id;

float et;

int tp;

int v;

int b;

};

int gcd(int a, int b){

if (b == 0)

return a;

return gcd(b, a % b);

}

int findlcm(int arr[], int n)

{

int ans = arr[0];

for (int i = 1; i < n; i++)

ans = (((arr[i] \* ans)) /(gcd(arr[i], ans)));

return ans;

}

void main(){

int n,ct=0,f=0;

float awt=0,atat=0,art=0,tp;

printf("Enter number of processes: ");

scanf("%d",&n);

struct P p[n];

struct P temp;

int a[n];

for(int i=0;i<n;i++){

p[i].id=i+1;

p[i].v=0;

printf("Enter Excecution Time and Time Period of P%d: ",i+1);

scanf("%f %d",&p[i].et,&p[i].tp);

p[i].b=p[i].et;

a[i]=p[i].tp;

}

for(int i=0;i<n;i++){

for(int j=i+1;j<n;j++){

if(p[i].tp>p[j].tp){

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

int ans=findlcm(a,n);

for(int i=0;i<ans;i++){

f=0;

for(int j = 0;j<n;j++){

if (i%p[j].tp==0) {

p[j].v=0;

p[j].et=p[j].b;

}

}

for(int j=0;j<n;j++){

if(p[j].v==0){

f=1;

p[j].et-=1;

printf("%d to %d P%d\n",i,i+1,p[j].id);

if(p[j].et==0){

p[j].v=1;

}

break;

}

}

if(f==0){

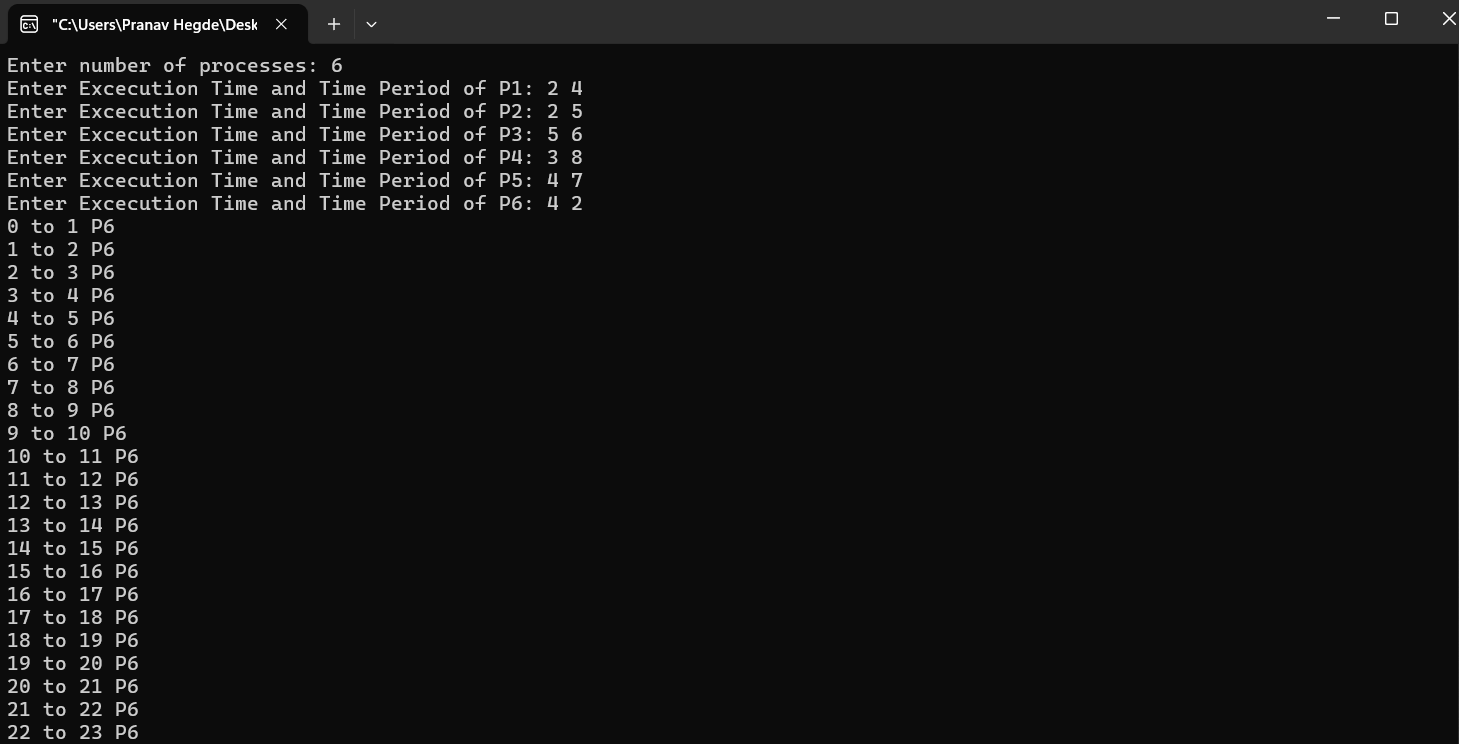
printf("%d to %d -\n",i,i+1);

}

}

}

**Output:**



1. **Earliest Deadline First**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_TSKS 10

typedef struct {

int p;

int c;

int d;

int rt;

int nd;

int id;

} Task;

void Input(Task tsks[], int \*n\_tsk) {

printf("Enter number of tasks (max %d): ", MAX\_TSKS);

scanf("%d", n\_tsk);

if (\*n\_tsk > MAX\_TSKS) {

printf("Number of tasks exceeds the maximum limit of %d.\n", MAX\_TSKS);

exit(EXIT\_FAILURE);

}

for (int i = 0; i < \*n\_tsk; i++) {

tsks[i].id = i + 1;

printf("Enter period (p) of task %d: ", i + 1);

scanf("%d", &tsks[i].p);

printf("Enter execution time (c) of task %d: ", i + 1);

scanf("%d", &tsks[i].c);

printf("Enter deadline (d) of task %d: ", i + 1);

scanf("%d", &tsks[i].d);

tsks[i].rt = tsks[i].c;

tsks[i].nd = tsks[i].d;

}

}

void EDF(Task tsks[], int n\_tsk, int tf) {

printf("\nEarliest-Deadline First Scheduling:\n");

for (int t = 0; t < tf; t++) {

int s\_tsk = -1;

for (int i = 0; i < n\_tsk; i++) {

if (t % tsks[i].p == 0) {

tsks[i].rt = tsks[i].c;

tsks[i].nd = t + tsks[i].d;

}

}

for (int i = 0; i < n\_tsk; i++) {

if (tsks[i].rt > 0 && (s\_tsk == -1 || tsks[i].nd < tsks[s\_tsk].nd)) {

s\_tsk = i;

}

}

if (s\_tsk != -1) {

printf("Time %d: Task %d\n", t, tsks[s\_tsk].id);

tsks[s\_tsk].rt--;

} else {

printf("Time %d: Idle\n", t);

}

}

}

int main() {

Task tsks[MAX\_TSKS];

int n\_tsk;

int tf;

Input(tsks, &n\_tsk);

printf("Enter time frame for simulation: ");

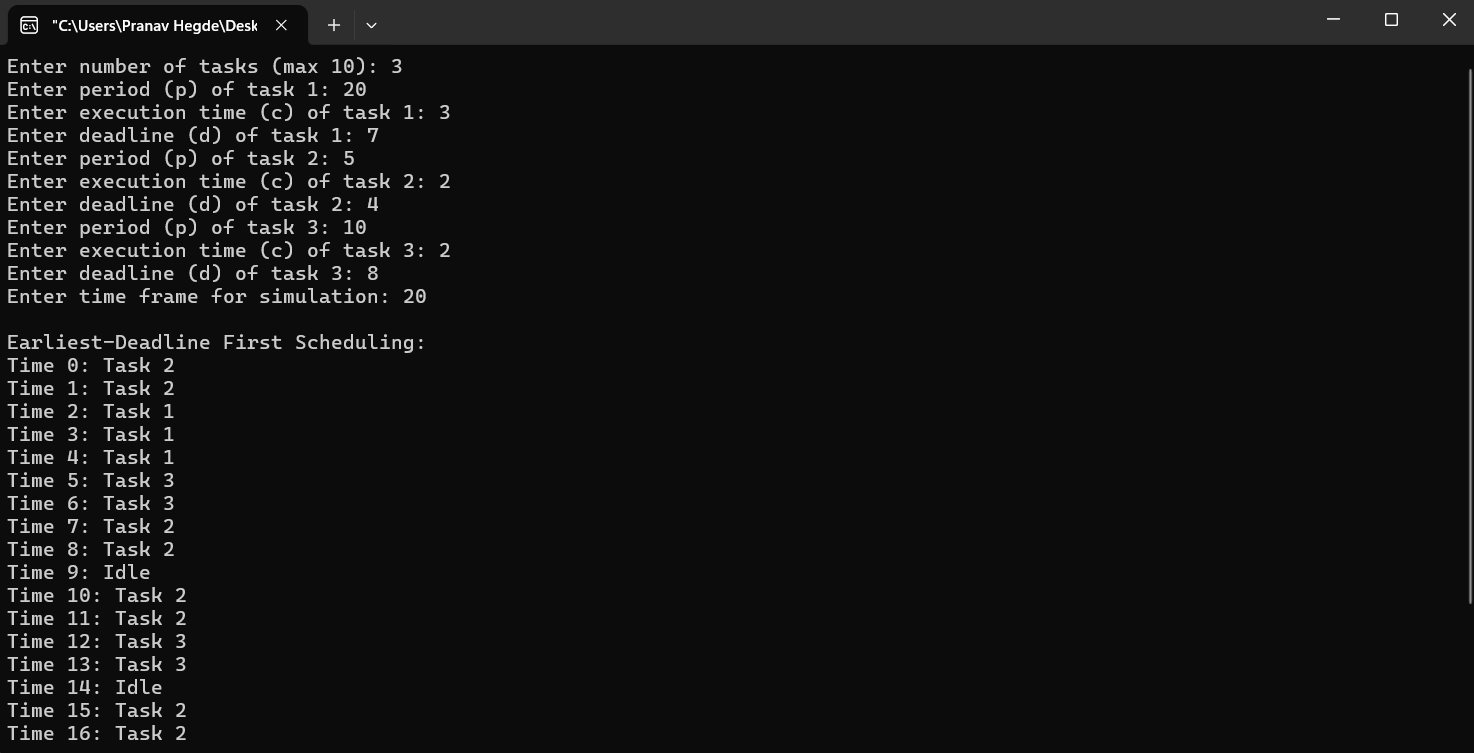
scanf("%d", &tf);

EDF(tsks, n\_tsk, tf);

return 0;

}

**Output:**



1. **Proportional Scheduling**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

int n, sOT = 0;

printf("Enter the number of processes: ");

scanf("%d", &n);

int pid[n];

int l[n + 1];

l[0] = 0;

printf("\nEnter the number of tickets for each process:\n");

for (int i = 0; i < n; i++) {

printf("PID%d: ", i + 1);

scanf("%d", &pid[i]);

sOT += pid[i];

l[i + 1] = pid[i];

}

int t = 1;

int sum = sOT;

for (int i = 0; i < n; i++) {

printf("Probability of servicing process %d is %d%%\n", i + 1, (pid[i] \* 100) / sOT);

}

srand(time(NULL));

while (sum > 0) {

int x = rand() % sOT;

int j;

for (j = 0; j < n; j++) {

if (x < l[j + 1]) {

printf("%d ms: Servicing Ticket of process %d\n", t, j + 1);

//l[j + 1]--;

pid[j]--;

sum--;

t++;

break;

}

}

}

for (int i = 0; i < n; i++) {

if (pid[i] == 0) {

printf("PID%d has finished executing\n", i + 1);

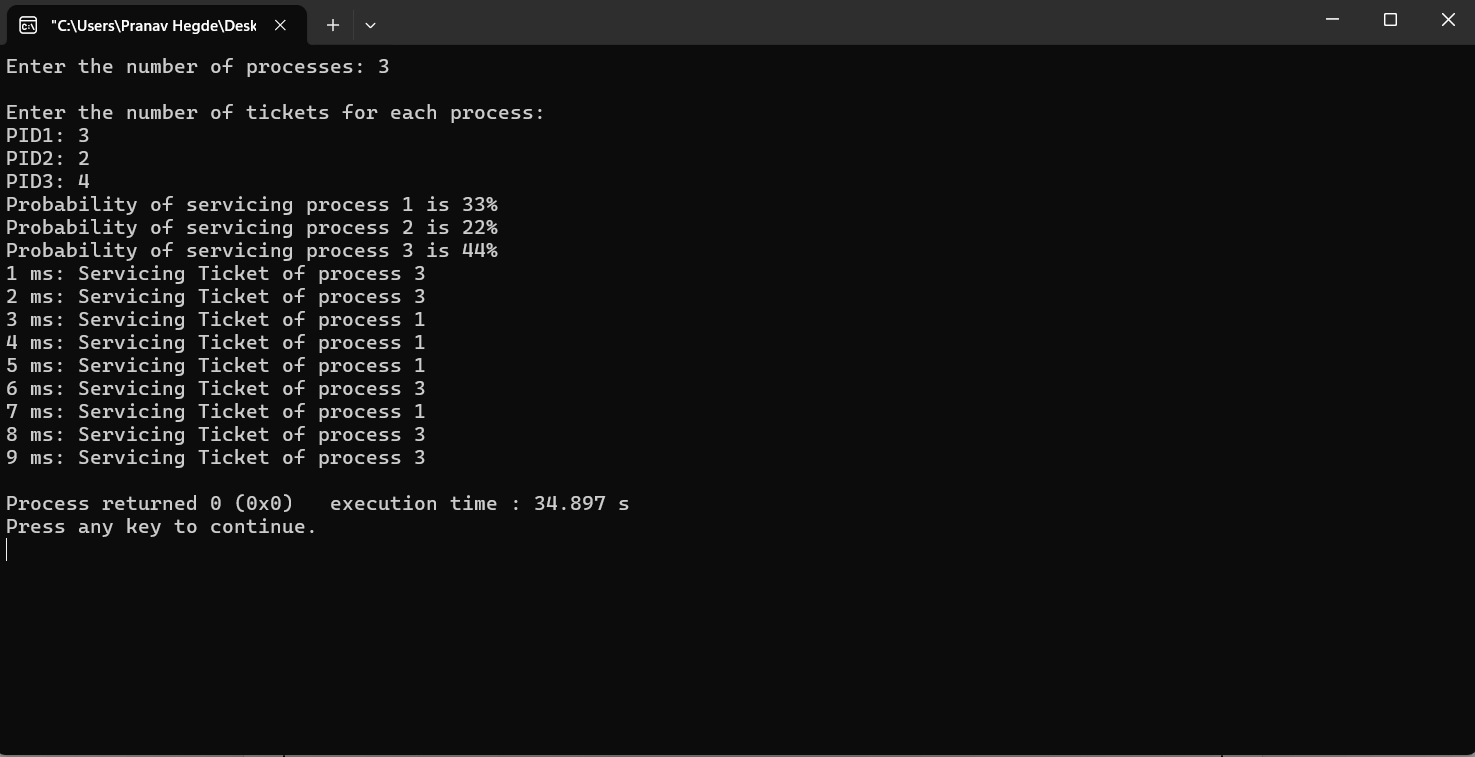
}

}

return 0;

}

**Output:**

****

**Lab Program - 6**

**Question:**

**Write a C program to simulate producer-consumer problem using semaphores.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#define BUF\_SIZE 10

#define MAX\_ITMS 20

int buf[BUF\_SIZE];

int cnt = 0;

int in = 0;

int out = 0;

int prod\_cnt = 0;

int cons\_cnt = 0;

pthread\_mutex\_t mtx;

pthread\_cond\_t cond\_prod;

pthread\_cond\_t cond\_cons;

void\* prod(void\* param) {

while (1) {

int item = rand() % 100;

pthread\_mutex\_lock(&mtx);

if (prod\_cnt >= MAX\_ITMS) {

pthread\_mutex\_unlock(&mtx);

break;

}

while (cnt == BUF\_SIZE) {

pthread\_cond\_wait(&cond\_prod, &mtx);

}

buf[in] = item;

in = (in + 1) % BUF\_SIZE;

cnt++;

prod\_cnt++;

printf("Produced: %d\n", item);

pthread\_cond\_signal(&cond\_cons);

pthread\_mutex\_unlock(&mtx);

sleep(rand() % 2);

}

return NULL;

}

void\* cons(void\* param) {

while (1) {

pthread\_mutex\_lock(&mtx);

if (cons\_cnt >= MAX\_ITMS) {

pthread\_mutex\_unlock(&mtx);

break;

}

while (cnt == 0) {

pthread\_cond\_wait(&cond\_cons, &mtx);

}

int item = buf[out];

out = (out + 1) % BUF\_SIZE;

cnt--;

cons\_cnt++;

printf("Consumed: %d\n", item);

pthread\_cond\_signal(&cond\_prod);

pthread\_mutex\_unlock(&mtx);

sleep(rand() % 2);

}

return NULL;

}

int main() {

pthread\_t tid\_prod, tid\_cons;

pthread\_mutex\_init(&mtx, NULL);

pthread\_cond\_init(&cond\_prod, NULL);

pthread\_cond\_init(&cond\_cons, NULL);

pthread\_create(&tid\_prod, NULL, prod, NULL);

pthread\_create(&tid\_cons, NULL, cons, NULL);

pthread\_join(tid\_prod, NULL);

pthread\_join(tid\_cons, NULL);

printf("Production & Consumption complete\n");

pthread\_mutex\_destroy(&mtx);

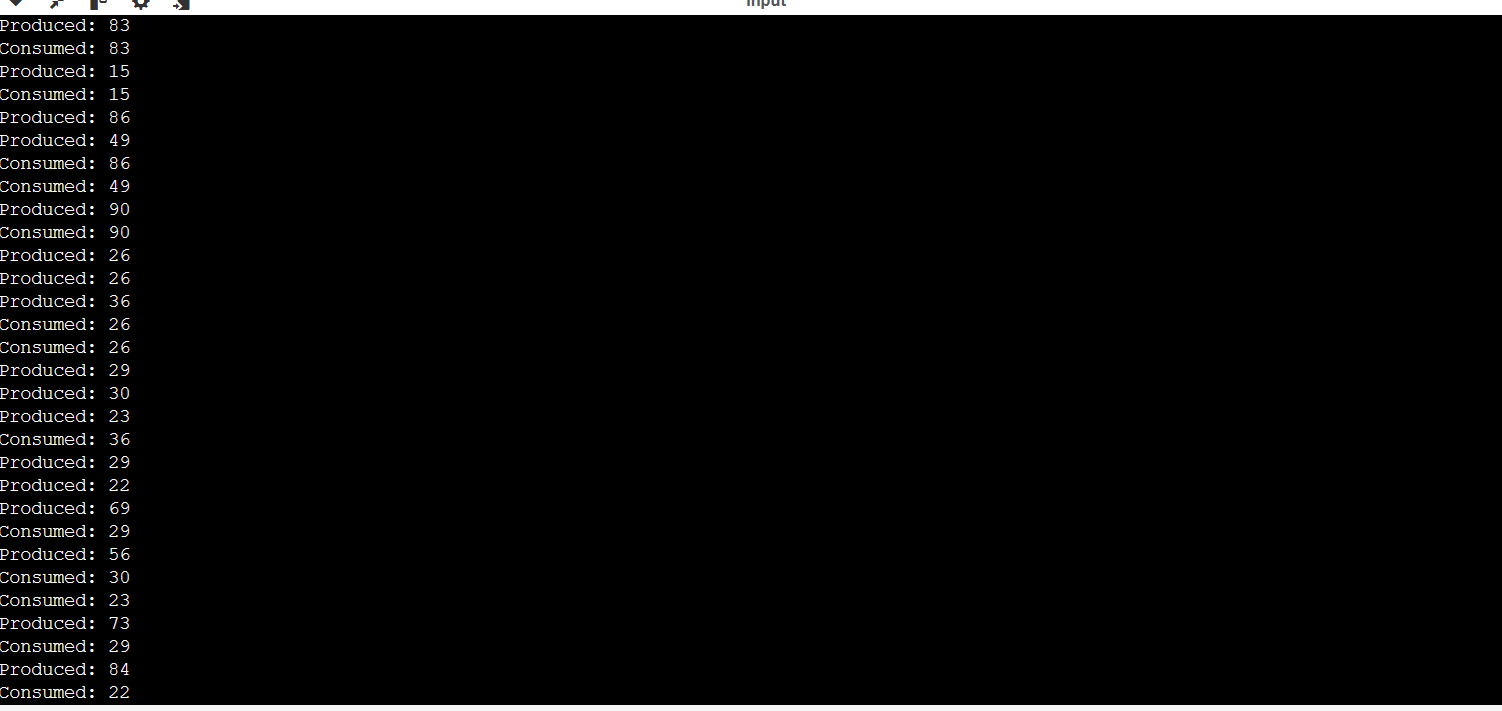
pthread\_cond\_destroy(&cond\_prod);

pthread\_cond\_destroy(&cond\_cons);

return 0;

}

**Output:**



**Lab Program - 7**

**Question:**

**Write a C program to simulate the concept of Dining-Philosophers problem.**

**Code:**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

for (i = 0; i < N; i++)

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philosopher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

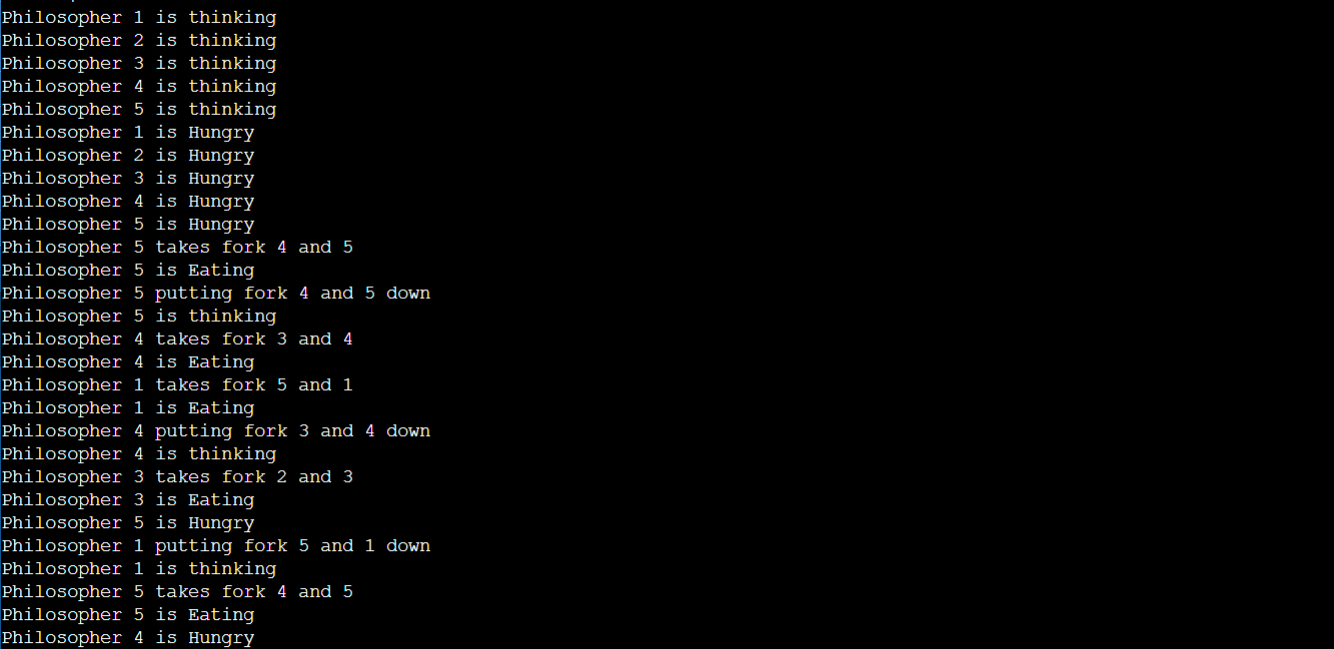
for (i = 0; i < N; i++)

pthread\_join(thread\_id[i], NULL);

}

**Output:**

**Dining-Philosopher**



**Lab Program - 8**

**Question:**

**Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance and detection.**

**Code:**

#include <stdio.h>

int main()

{

int n, m, i, j, k;

n = 5;

m = 3;

int alloc[5][3] = { { 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int avail[3] = { 3, 3, 2 };

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][m];

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

for (i = 0; i < n; i++) {

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

for(int i=0;i<n;i++)

{

if(f[i]==0)

{

flag=0;

printf("The following system is not safe");

break;

}

}

if(flag==1)

{

printf("Following is the SAFE Sequence\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

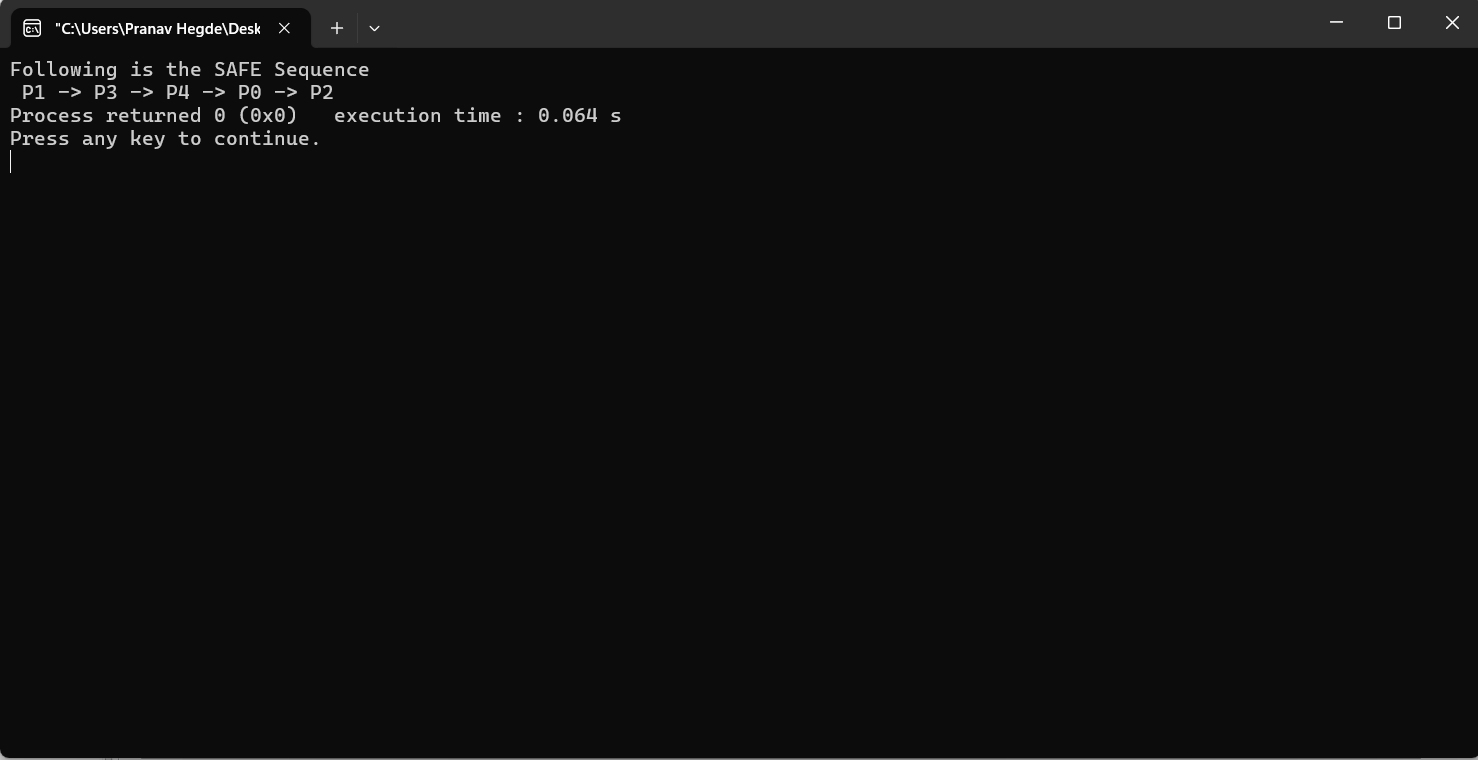
printf(" P%d", ans[n - 1]);

}

return (0);

}

**Output:**



**Lab Program - 9**

**Question:**

**Write a C program to simulate the following contiguous memory allocation techniques   
a) Worst-fit   
b) Best-fit   
c) First-fit**

**Code:**

#include <stdio.h>

struct Block {

int block\_no;

int block\_size;

int is\_free;

};

struct File {

int file\_no;

int file\_size;

};

void firstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("Memory Management Scheme - First Fit\n");

printf("File\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment\n");

for (int i = 0; i < n\_files; i++) {

for (int j = 0; j < n\_blocks; j++) {

if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {

blocks[j].is\_free = 0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", files[i].file\_no, files[i].file\_size, blocks[j].block\_no, blocks[j].block\_size, blocks[j].block\_size - files[i].file\_size);

break;

}

}

}

}

void worstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("Memory Management Scheme - Worst Fit\n");

printf("File\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment\n");

for (int i = 0; i < n\_files; i++) {

int worst\_fit\_block = -1;

int max\_fragment = -1;

for (int j = 0; j < n\_blocks; j++) {

if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {

int fragment = blocks[j].block\_size - files[i].file\_size;

if (fragment > max\_fragment) {

max\_fragment = fragment;

worst\_fit\_block = j;

}

}

}

if (worst\_fit\_block != -1) {

blocks[worst\_fit\_block].is\_free = 0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", files[i].file\_no, files[i].file\_size, blocks[worst\_fit\_block].block\_no, blocks[worst\_fit\_block].block\_size, max\_fragment);

}

}

}

void bestFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("Memory Management Scheme - Best Fit\n");

printf("File\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment\n");

for (int i = 0; i < n\_files; i++) {

int best\_fit\_block = -1;

int min\_fragment = 10000;

for (int j = 0; j < n\_blocks; j++) {

if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {

int fragment = blocks[j].block\_size - files[i].file\_size;

if (fragment < min\_fragment) {

min\_fragment = fragment;

best\_fit\_block = j;

}

}

}

if (best\_fit\_block != -1) {

blocks[best\_fit\_block].is\_free = 0;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", files[i].file\_no, files[i].file\_size, blocks[best\_fit\_block].block\_no, blocks[best\_fit\_block].block\_size, min\_fragment);

}

}

}

int main() {

int n\_blocks, n\_files;

printf("Enter the number of blocks: ");

scanf("%d", &n\_blocks);

printf("Enter the number of files: ");

scanf("%d", &n\_files);

struct Block blocks[n\_blocks];

for (int i = 0; i < n\_blocks; i++) {

blocks[i].block\_no = i + 1;

printf("Enter the size of block %d: ", i + 1);

scanf("%d", &blocks[i].block\_size);

blocks[i].is\_free = 1;

}

struct File files[n\_files];

for (int i = 0; i < n\_files; i++) {

files[i].file\_no = i + 1;

printf("Enter the size of file %d: ", i + 1);

scanf("%d", &files[i].file\_size);

}

firstFit(blocks, n\_blocks, files, n\_files);

printf("\n");

for (int i = 0; i < n\_blocks; i++) {

blocks[i].is\_free = 1;

}

worstFit(blocks, n\_blocks, files, n\_files);

printf("\n");

for (int i = 0; i < n\_blocks; i++) {

blocks[i].is\_free = 1;

}

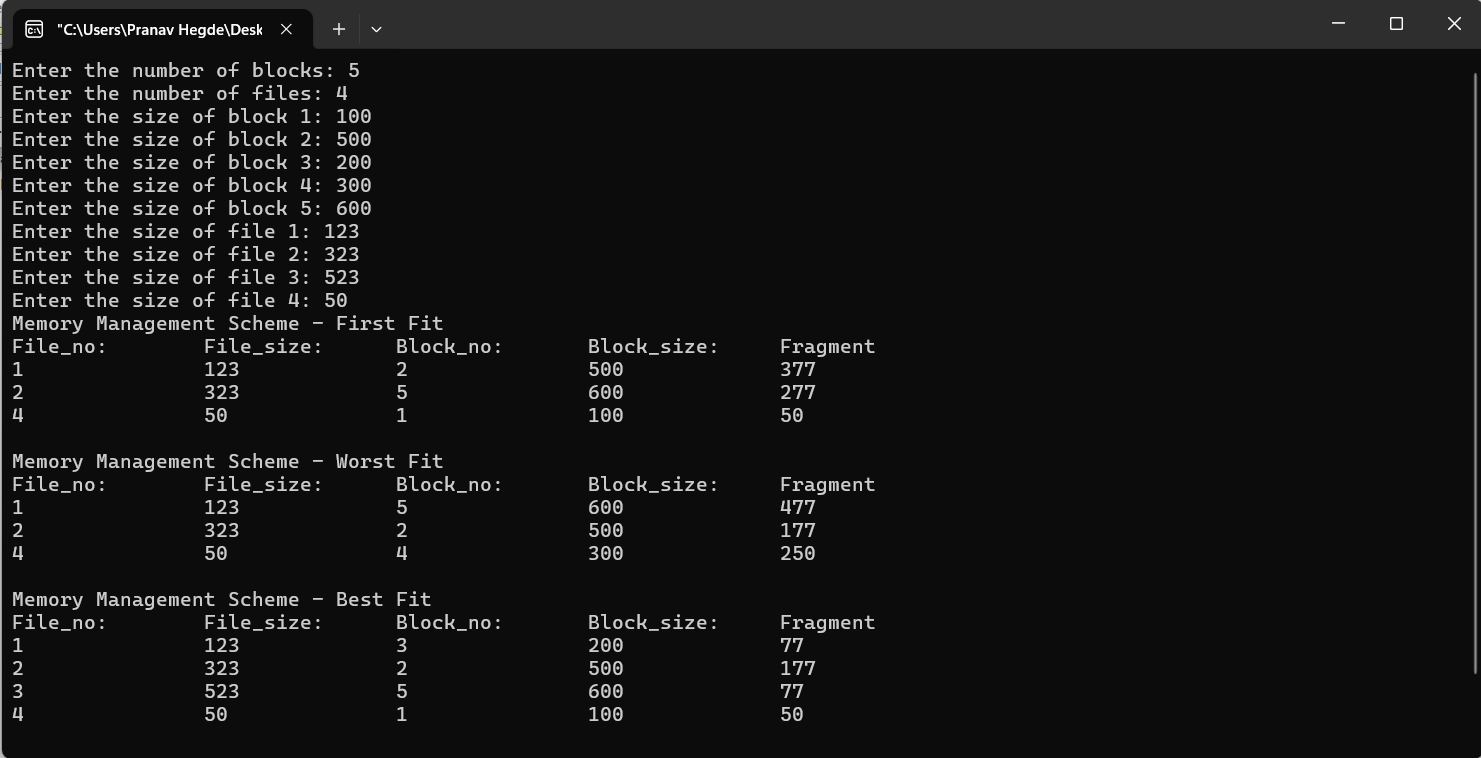
bestFit(blocks, n\_blocks, files, n\_files);

return 0;

}

**Output:**

**Memory Allocation (First Fit, Best Fit, Worst Fit)**

****

**Lab Program - 10**

**Question:**

**Please execute the page Replacement Algorithms: FIFO, OPTIMAL and LRU**

**Code:**

#include <stdio.h>

int isPagePresent(int frames[], int n, int page) {

for (int i = 0; i < n; i++) {

if (frames[i] == page) {

return 1;

}

}

return 0;

}

void printFrames(int frames[], int n) {

for (int i = 0; i < n; i++) {

if (frames[i] != -1) {

printf("%d ", frames[i]);

} else {

printf("- ");

}

}

printf("\n");

}

void fifoPageReplacement(int pages[], int numPages, int numFrames) {

int frames[numFrames];

int front = 0, pageFaults = 0;

for (int i = 0; i < numFrames; i++) {

frames[i] = -1;

}

printf("FIFO Replacement\n");

printf("Reference String\tFrames\n");

for (int i = 0; i < numPages; i++) {

printf("%d\t\t", pages[i]);

if (!isPagePresent(frames, numFrames, pages[i])) {

frames[front] = pages[i];

front = (front + 1) % numFrames;

pageFaults++;

}

printFrames(frames, numFrames);

}

printf("\nTotal Page Faults: %d\n\n", pageFaults);

}

int findOptimalReplacementIndex(int pages[], int numPages, int frames[], int numFrames, int currentIndex) {

int farthest = currentIndex;

int index = -1;

for (int i = 0; i < numFrames; i++) {

int j;

for (j = currentIndex; j < numPages; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

index = i;

}

break;

}

}

if (j == numPages) {

return i;

}

}

return (index == -1) ? 0 : index;

}

void optPageReplacement(int pages[], int numPages, int numFrames) {

int frames[numFrames];

int pageFaults

for (int i = 0; i < numFrames; i++) {

frames[i] = -1;

}

printf("Optimal Replacement\n");

printf("Reference String\tFrames\n");

for (int i = 0; i < numPages; i++) {

printf("%d\t\t", pages[i]);

if (!isPagePresent(frames, numFrames, pages[i])) {

if (isPagePresent(frames, numFrames, -1)) {

for (int j = 0; j < numFrames; j++) {

if (frames[j] == -1) {

frames[j] = pages[i];

break;

}

}

} else {

int index = findOptimalReplacementIndex(pages, numPages, frames, numFrames, i + 1);

frames[index] = pages[i];

}

pageFaults++;

}

printFrames(frames, numFrames);

}

printf("\nTotal Page Faults: %d\n\n", pageFaults);

}

void lruPageReplacement(int pages[], int numPages, int numFrames) {

int frames[numFrames];

int pageFaults = 0;

int timestamps[numFrames];

for (int i = 0; i < numFrames; i++) {

frames[i] = -1;

timestamps[i] = -1;

}

printf("LRU Replacement\n");

printf("Reference String\tFrames\n");

for (int i = 0; i < numPages; i++) {

printf("%d\t\t", pages[i]);

if (!isPagePresent(frames, numFrames, pages[i])) {

int lruIndex = 0;

for (int j = 1; j < numFrames; j++) {

if (timestamps[j] < timestamps[lruIndex]) {

lruIndex = j;

}

}

frames[lruIndex] = pages[i];

timestamps[lruIndex] = i;

pageFaults++;

} else {

for (int j = 0; j < numFrames; j++) {

if (frames[j] == pages[i]) {

timestamps[j] = i;

break;

}

}

}

printFrames(frames, numFrames);

}

printf("\nTotal Page Faults: %d\n\n", pageFaults);

}

int main() {

int numFrames, numPages;

printf("Enter the number of frames: ");

scanf("%d", &numFrames);

printf("Enter the number of pages: ");

scanf("%d", &numPages);

int pages[numPages];

printf("Enter the reference string: ");

for (int i = 0; i < numPages; i++) {

scanf("%d", &pages[i]);

}

fifoPageReplacement(pages, numPages, numFrames);

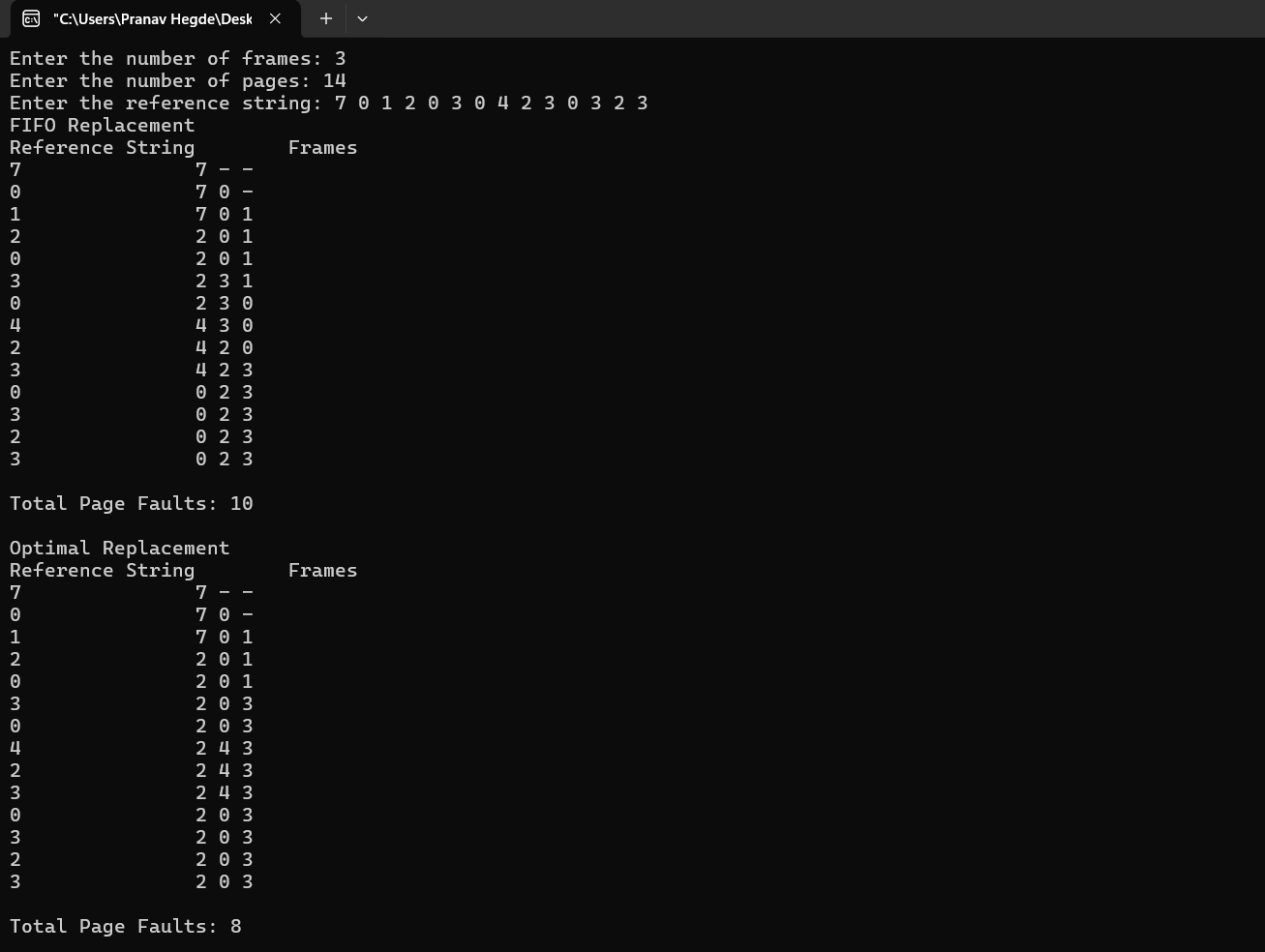
optPageReplacement(pages, numPages, numFrames);

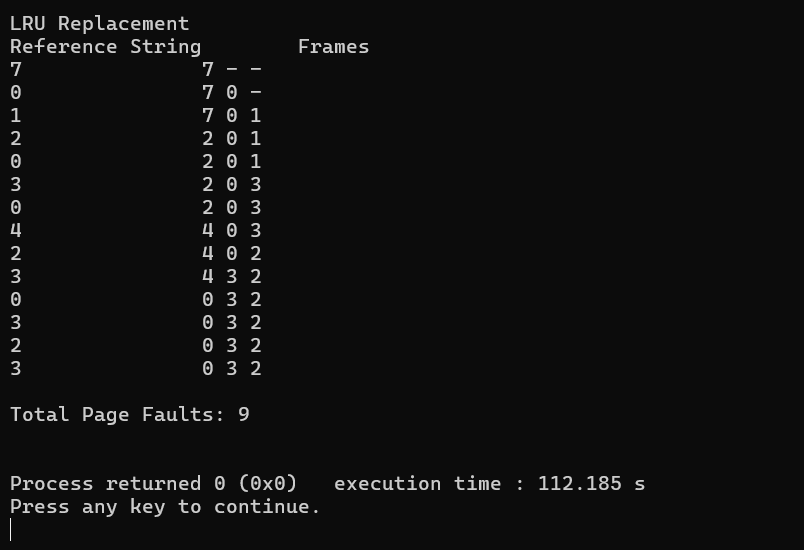
lruPageReplacement(pages, numPages, numFrames);

return 0;

}

**Output:**

****

****